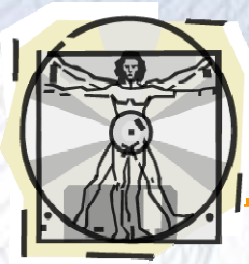
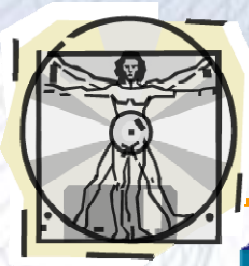
SENG 697
Agent-based Software Engineering

Behrouz Far
Schulich School of Engineering, University of Calgary
far@ucalgary.ca
http://www.enel.ucalgary.ca/People/far/



Contents

- Introduction
- Software vs. software agent
- Software agent models
- Agent decision making process
- Conclusion



Course Curriculum

Overview of agent-based SE

Principles of Object Technology

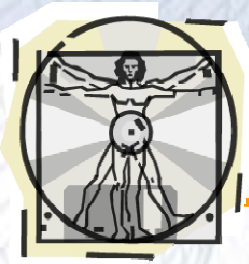
Methodologies for agent-based analysis and design

Agent communication & knowledge sharing

Agent-based System Architecture & Organization

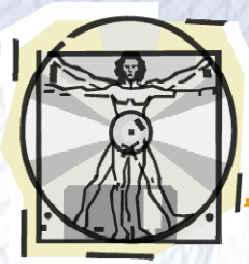
FIPA: Foundation for Intelligent Physical Agents

Other topics:
Agent Interaction,
Infrastructure, APIs,
Performance metrics,
Learning,
Self-organizing systems
etc.



Realities: High Risk

- Software development is a very high risk task (may be!)
- About 20% of the software projects are canceled
- About 84% of software projects are incomplete when released (need patches, service packs, etc.)
- Almost all of the software projects costs exceed initial estimations (cost overrun)
- **Question:** What can be done to improve **software product** and **software development process**?

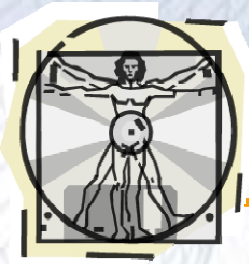


Software Product: Trends …

- **Past:** centralized computing system; monolithic programming model
- **Now:** distributed computing system; heterogeneous, scalable, open and distributed programming model

Nowadays, an increasing number of software projects are revised, restructured and reconstructed in terms of *software agents*: MAS

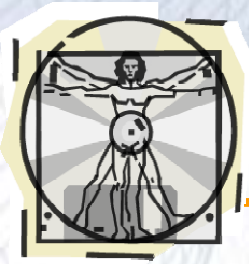
"It's the latest innovation in office safety. When your computer crashes, an air bag is activated so you won't bang your head in frustration."



Software Process: Trends …

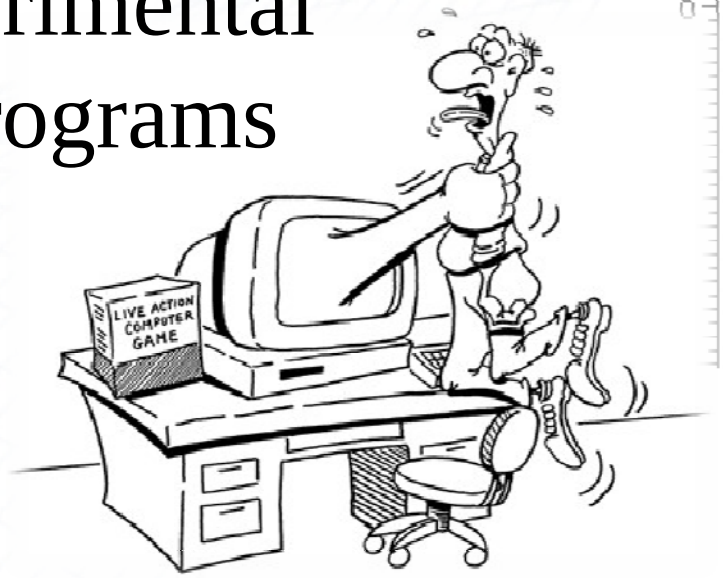
- **Past:** waterfall development process model
- **Now:** incremental, agile, experimental development process model

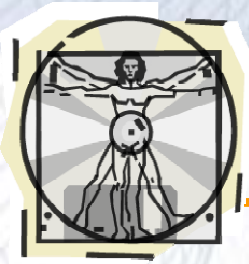
- **Agent-based software development:** new way of analysis and synthesis of software systems (supposed to be …)



Paradigm Shift …

- May be the "software concept" itself should be revised not only the process and/or the product
- **Software agents:** new experimental embodiment of computer programs





Controversial Definitions /1

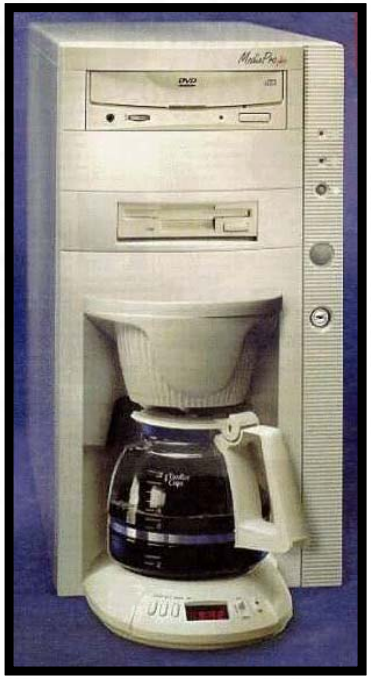
- An agent is a **software entity** that functions continuously and autonomously in a particular environment, often inhabited by other agents and processes [Shoham 97]

Possible examples:

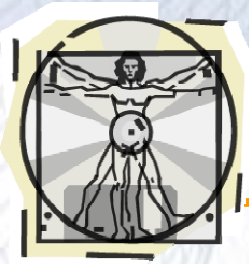
A cell phone?

A time controlled sprinkler?

A notification program: e.g., CCTray?



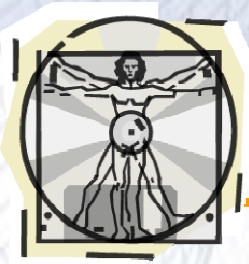
- Other concerns such as weight, knowledge, learning, inference capability, etc. … are equally important



Controversial Definitions /2

An agent is a software entity that might possess some of the following attributes

- **Reactivity** (i.e., selectively sense and act)
- **Autonomy** (i.e., goal directness, proactive & self-started behavior)
- **Collaborative** (i.e., work with other agents and entities to achieve a common goal)
- **Knowledge-level communication ability** (i.e., communicate with other entities in a language like speech-act, higher level than symbol level program to program protocols)
- **Inferential capability** (i.e., act on abstract task spec., using models of self, situation, and/or other agents)
- **Temporal continuity** (i.e., persistence of state and personality)
- **Personality** (i.e., manifesting attributes of a believable agent. Personality is composed of beliefs and behavior based on those beliefs)
- **Adaptability** (i.e., learn and improve with experience)
- **Mobility** (i.e., migrate from one host to another in a self-directed way)



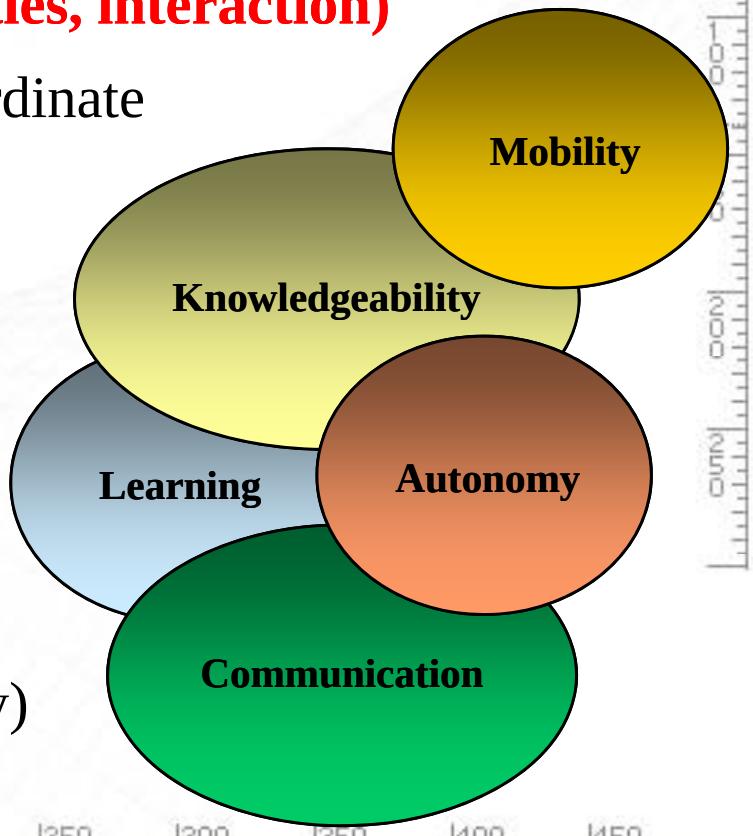
Controversial Definitions /3

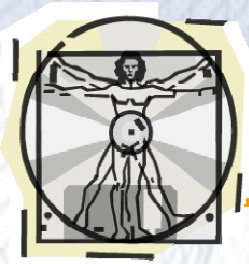
An agent is a software entity that exhibits (a subset of) the following characteristics:

- **Knowledgeability (cognitive capabilities, interaction)**

  Capability to interact (cooperate, coordinate and compete) with the other people, agents and processes
- **Learning**

  Capability to learn autonomously
- **Autonomy** (set own goal, proactive)
- **Communication** (high level language)
- **Mobility** (physical or software mobility)



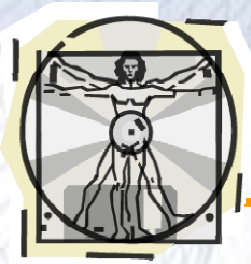


Controversial Development

- Agent-oriented programming
- Agent-oriented software development
- Agent-oriented design

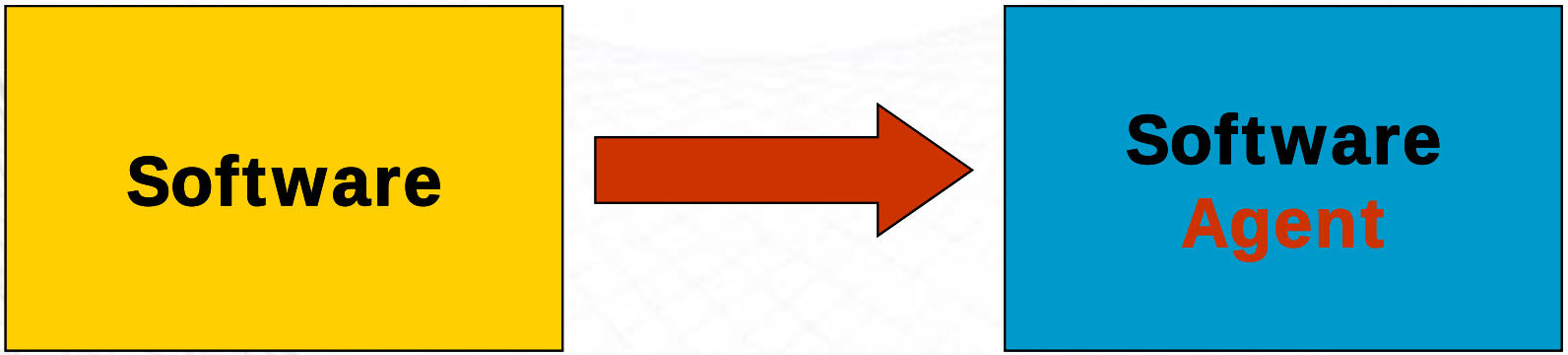
etc.

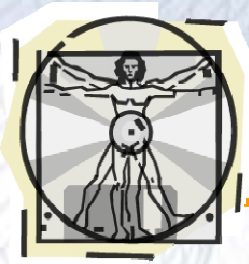
Agent system development is dominated by informal guidelines, heuristics and inspirations rather than formal principles and well-defined engineering techniques. [Jennings] Good on paper… hard to use in practice



Software vs. Software Agent

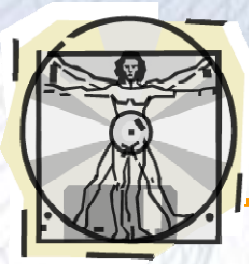
- Similarities and differences:
  - **Software** aspect of a software agent
  - **Agent** aspect of a software agent





What Makes them Different?

1. Roles vs. tasks
2. Emergent behaviour
3. Knowledge cycle
4. Interactions vs. intra-actions
5. Symbol level communication
6. Knowledge completeness
7. Indeterminism: Decision making



1. Roles vs. Tasks

Use Cases: How to use bathroom

- We assign "**tasks**" to software

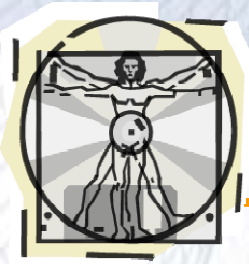
  Example: on-line registration software

  - "What" and "how" are specified in advance (use cases, scenarios, etc.)
  - Changes in requirements are not tolerable

- We assign "**roles**" to software agents

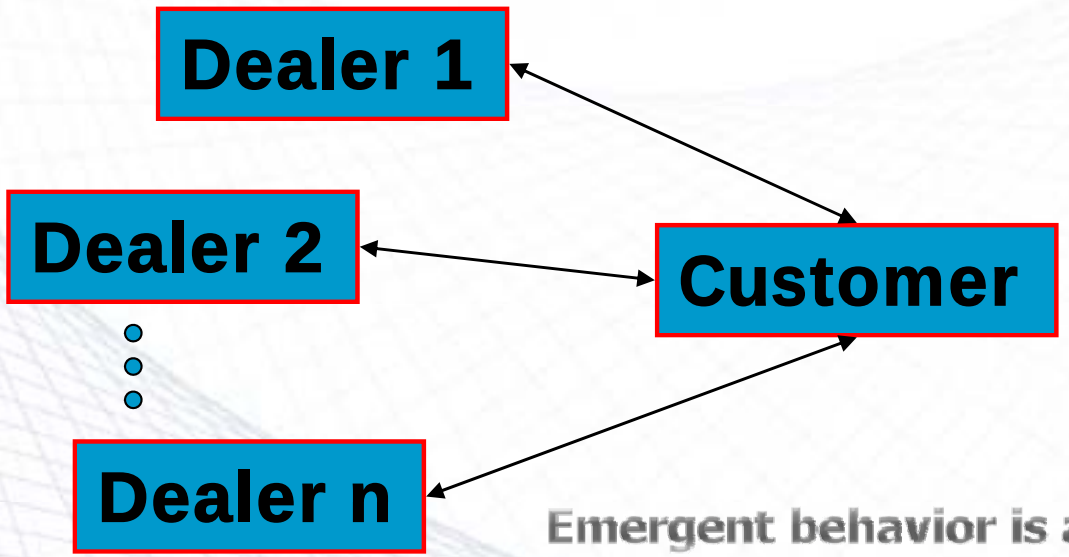
  Example: registration agent; travel agent; secretary

  - "What" is specified in advance. "How" is determined dynamically ← **"how to" library**
  - Changes in requirement can be tolerated



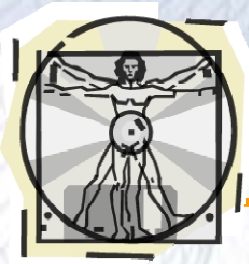
2. Emergent Behaviour

- Agent system behaviour is not “fully predefined”. It is the result of dynamic interaction among the participants
- Example: Electronic Commerce System



Dealer agents learnt that they can maximize their profit by sharing the invoice price among themselves!

Emergent behavior is appreciated not avoided



3. Knowledge Cycle

- Data ⇒ Information ⇒ Knowledge
  - ***Data*** is a sequence of quantified or quantifiable symbols
  - ***Information*** is about taking data and putting it into a meaningful pattern
  - ***Knowledge*** is the ability to use that information

- ***Intelligence*** ⇒ *Data* ⇒ *Information* ⇒ *Knowledge*

- How intelligence can be gathered?
  - Informants, consultants, experts
  - Yellow pages, Web, external signals, knowledge-bases

"Thinking outside of the box is difficult for some people. Keep trying."

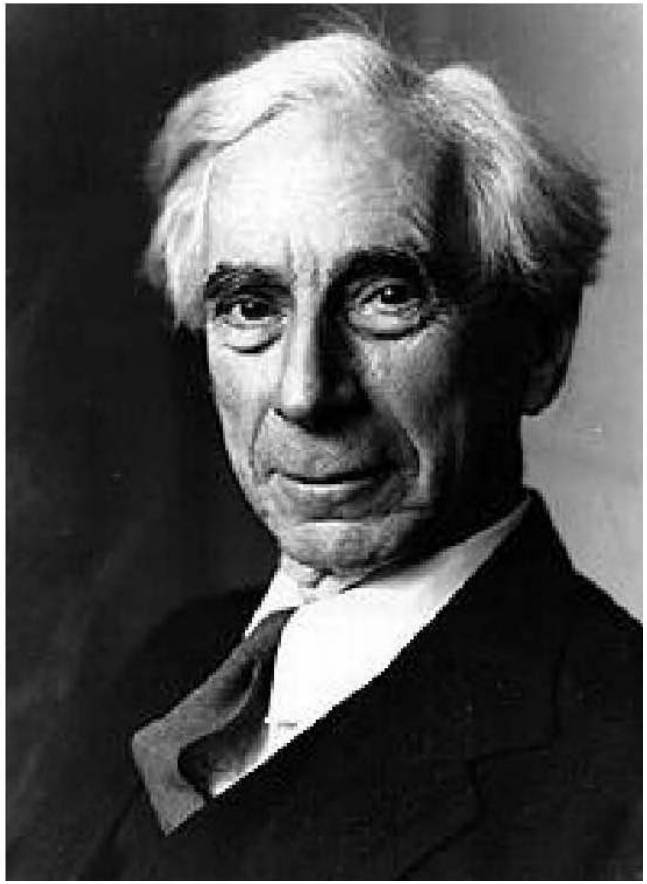
Knowledge Cycle (cont'd)

Theory of Knowledge

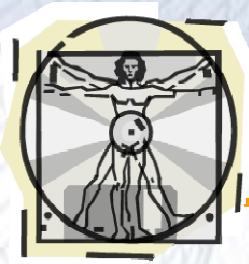
(Encyclopaedia Britannica)

Concepts:

Knowledge, belief, words, personality, behaviour, truth, uncertainty, data, inference, probability, etc.

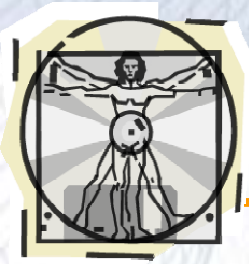


Bertrand Russell



4. Interactions vs. Intra-actions

- In software systems, relationships among subsystems (or components, packages) are:
  - ***Inter-actions*** among subsystems and
  - ***Intra-actions*** within a subsystem
- ***Interactions*** are between the artifact and its outer environment. E.g., giveaway: give the puck to the other team
- ***Intra-actions*** are the characteristics of the artifact's inner environment. E.g., pass: pass the puck to a teammate



4. Interactions vs. Intra-actions

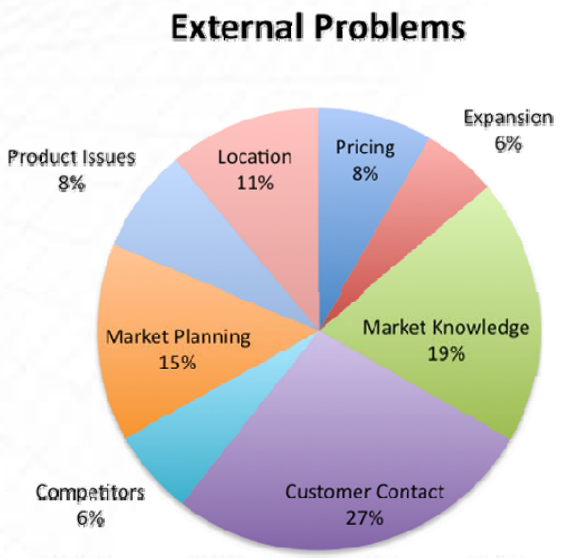
- In nearly decomposable systems *intra-actions* are more frequent (typically at least 10 times more) and more predictable than *inter-actions* [Herbert Simon].
- Software systems are nearly decomposable: subsystems can be treated almost independently.
- Although many of the *inter-actions* can be predictable at design time, some simply cannot.

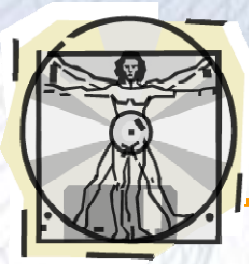
Therefore

- Conventional software engineering approaches can handle well *intra-actions* within or among components.
- Agent-approach is needed to handle *inter-actions* among (clusters of) software components. ← **focus on architecture**

Example: Inter vs. Intra-actions

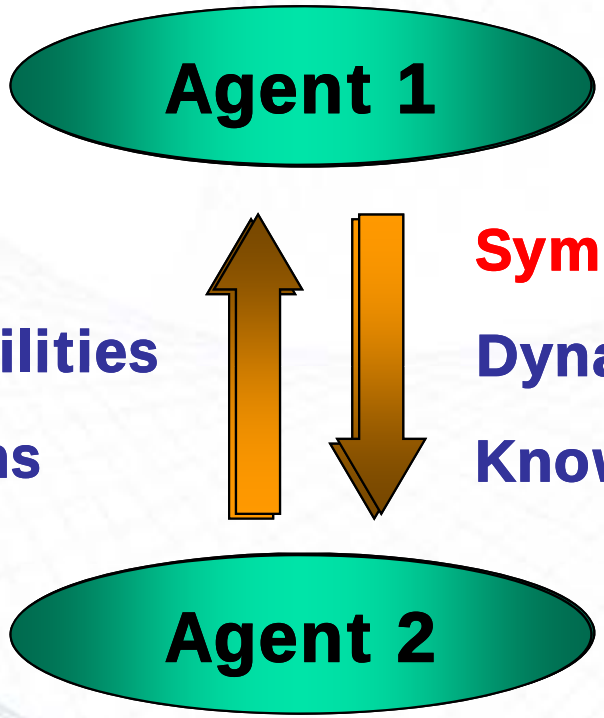
- Internal and External Problems Experienced by Entrepreneurs

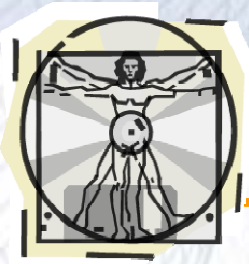




5. Symbol Level Communication

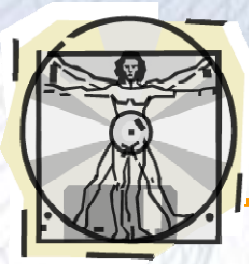
- **Software: Signal-level communication**
- **Agents: Symbol-level communication**



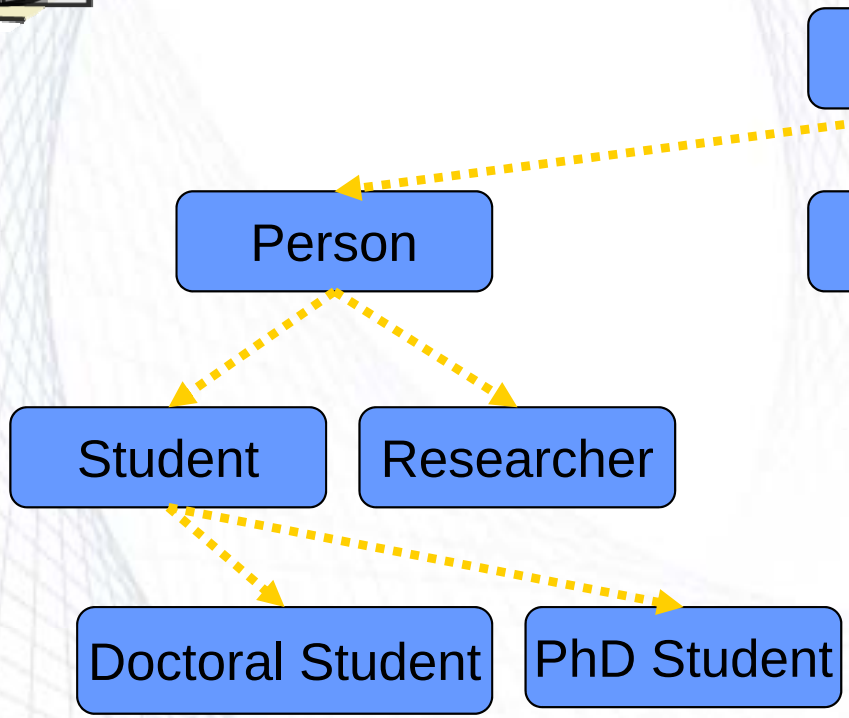


5. Symbol Level Communication

- For a message from one agent to be understood by another, the agents must ascribe the same meaning to the constants used in the message
- Thus, we require an "**ontology**" to map a given constant to some well-understood meaning
- **Dynamic Message Passing:**
  - Agents should first discover whether or not they share a mutual understanding of the domain constants before further message passing

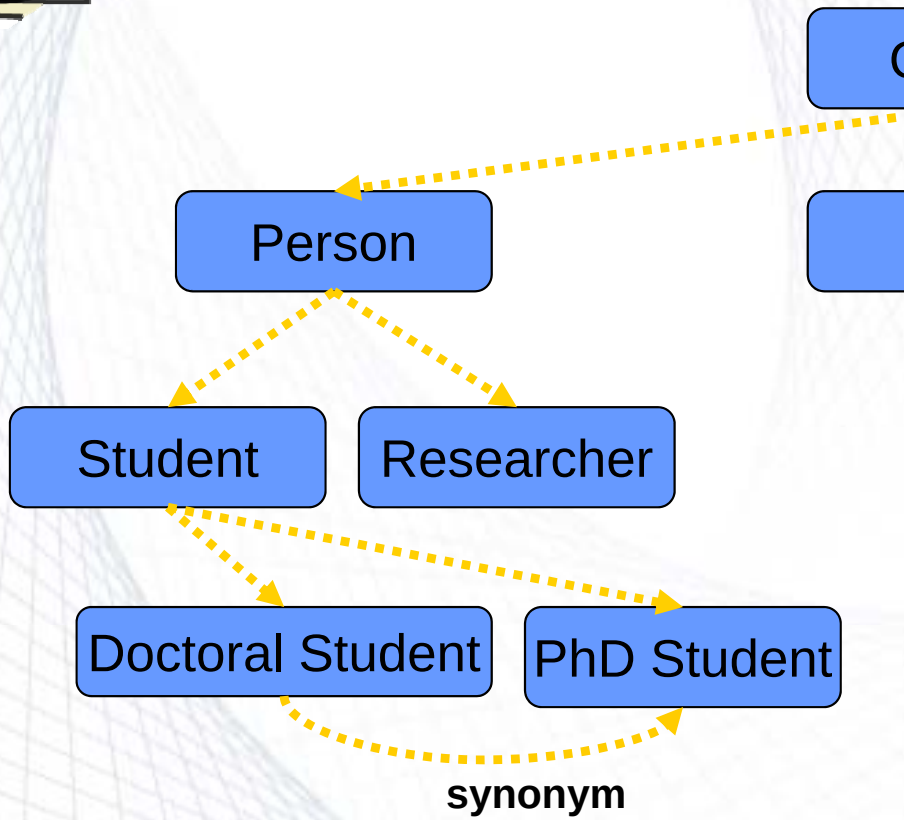


Taxonomy: What is?

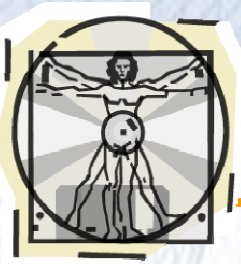


Taxonomy := Segementation, classification and ordering of elements into a classification system according to the relationships among them

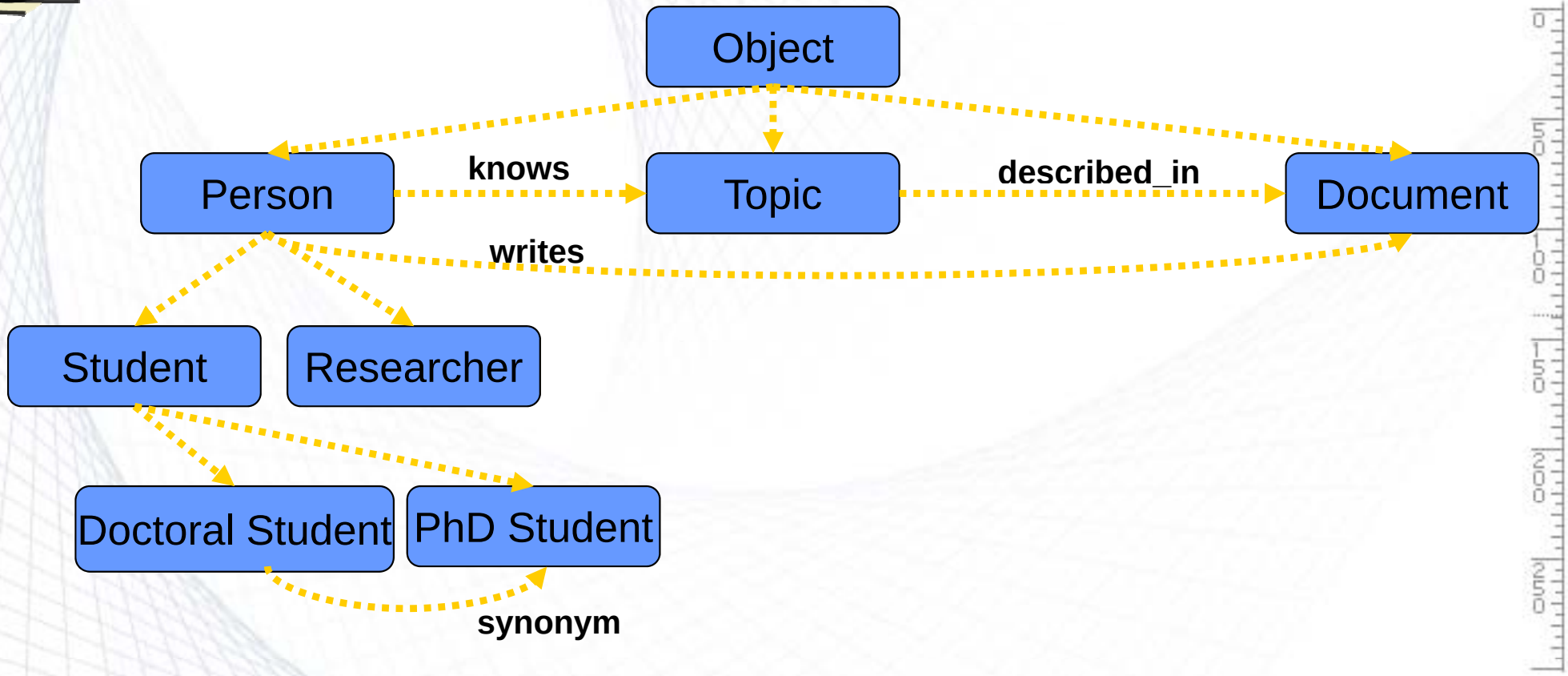
Thesaurus: What is?



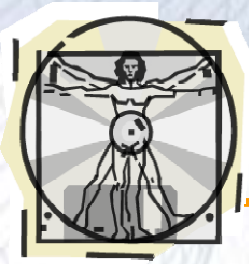
- Terminology for specific domain
- Graph with primitives, 2 fixed relationships (similar, synonym)
- Originated from bibliography



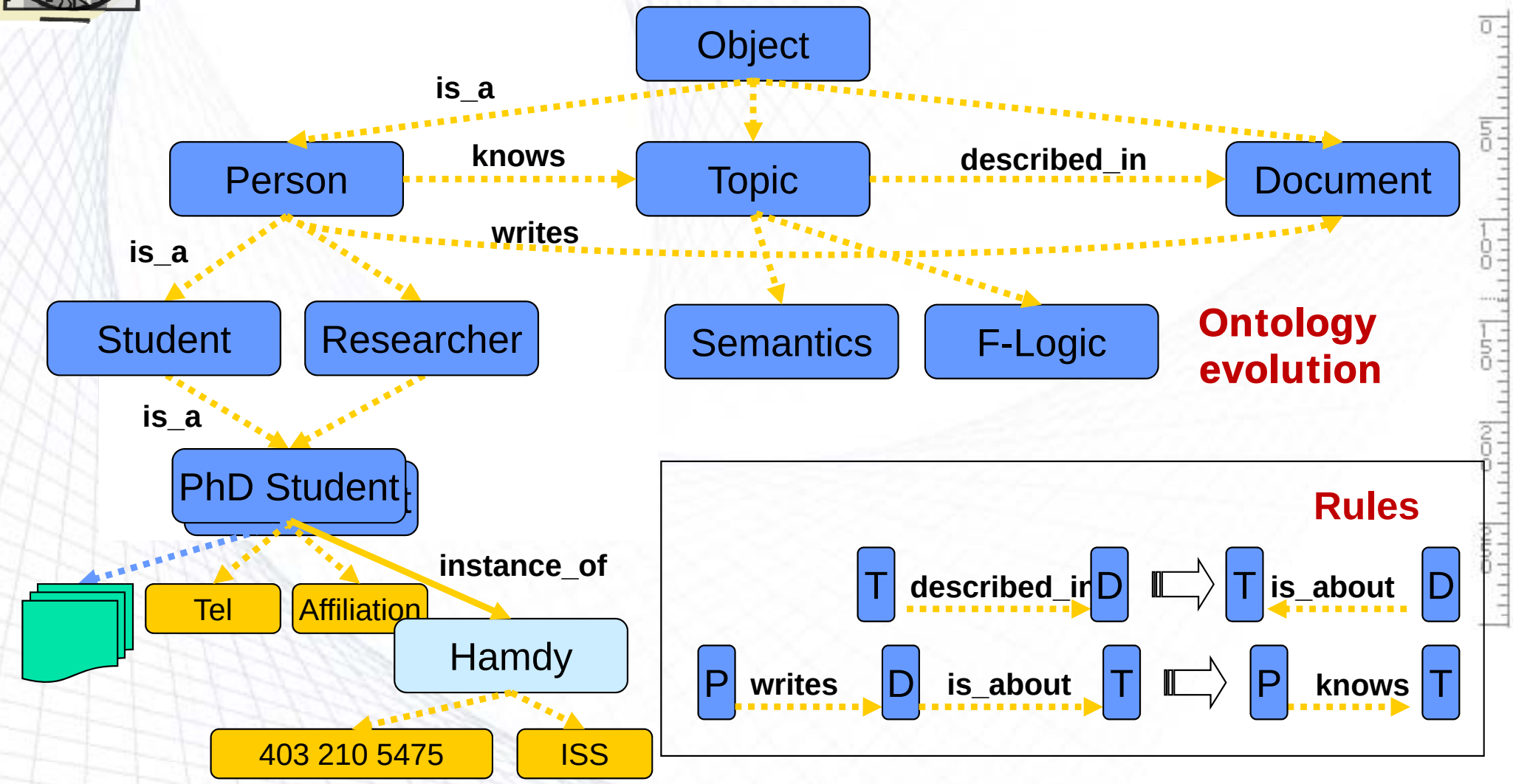
Topic Map: What is?

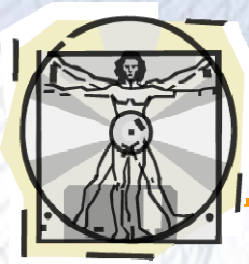


- Topics (nodes), relationships and o*ccurences* (to documents)
- Typically for navigation- and visualisation



Ontology: What is?

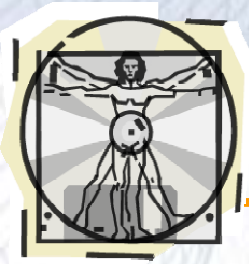




Ontology

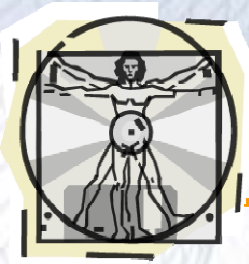
- Not only conceptualization but also relations
- May be no good for human but good for machine processing

"Now! ... *That* should clear up a few things around here!"



6. Knowledge Completeness

- Interacting software systems ***must*** have complete knowledge of each other
- Interacting software agents ***may*** have complete knowledge about the other agents' goals, strategies (i.e., actions to select from) and utilities (i.e., the pay-offs of actions)
- Interactions based on the complete knowledge assumption are usually classified as ***cooperation*** and ***coordination***
- In many cases knowledge completeness assumption may not hold: ***competition***

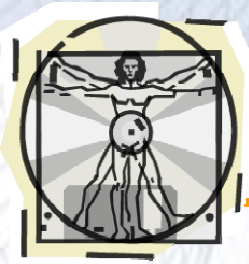


7. Indeterminism: Decision Making

- **Software systems:** Decision points in algorithms are deterministic
- **"Intelligent" software systems:** empowered by *Reasoning* mechanisms
- ***Reasoning*** is based on a single thread of control

  ← has long term consequences
- ***Decision making*** is based on multiple threads of control!

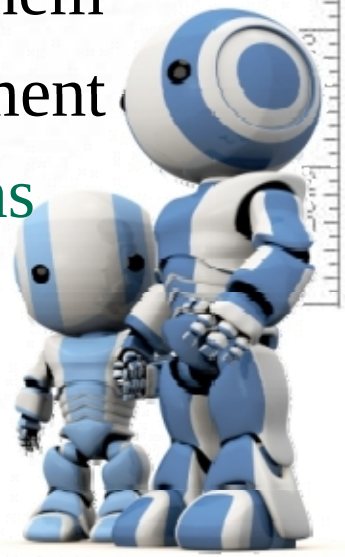
  ← has immediate consequences

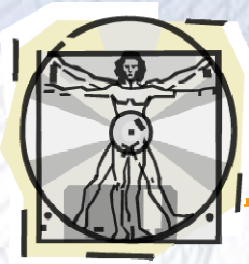


Therefore: Definition

- Software agents are
  - ***Knowledgeable:*** capable of managing their own set of beliefs (information) and/or desires (goals) and they can decide upon the next operations to execute
  - ***Autonomous:*** we don't define tasks for them instead we assign roles and responsibilities to them
  - ***Situated:*** defined with respect to their environment
  - ***Interactive:*** capable of filtering communications and managing dialogues

  software entities.





Agent System Development

- Analogy:
  - Software agent vs. “movie actor” playing “roles”
  - Use-cases vs. “screen play” or “scenario”
  - Software “designer” and “engineer” vs. “producer” and “director”
- Agent development methodologies:
  - 30+ available
  - 6+ Frequently used